

AD-A172 757 METHODS OF MULTIPLE ACCESS COMMUNICATIONS WITH ENERGY  
DETECTORS(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB  
FOR INFORMATION AND D R D YATES MAY 86 LIDS-TH-1557

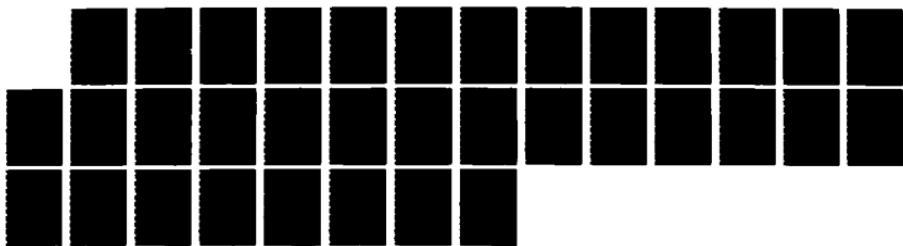
1/1

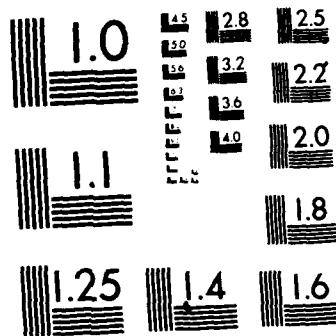
UNCLASSIFIED

N00814-84-K-0357

F/G 17/2

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A172 757

May 1986

LIDS-TH-1557

MR-2 77. 330

METHODS OF MULTIPLE ACCESS COMMUNICATIONS  
WITH ENERGY DETECTORS

by

Roy D. Yates

This report is based on the unaltered thesis of Roy D. Yates, submitted in partial fulfillment of the requirements for the degree of Master of Science at the Massachusetts Institute of Technology, Laboratory for Information and Decision Systems with partial support provided by the Defense Advanced Research Projects Agency under contract No. ONR/N00014-84-K-0357, the National Science Foundation under grant NSF-ECS-8310698 and the Army Research Office grant DAAG29-84-K-0005



CLEARED  
FOR OPEN PUBLICATION

SEP 23 1986 21

DIRECT UNCLASSIFIED INFORMATION  
AND SECURITY REVIEW (OASD-PA)  
DEPARTMENT OF DEFENSE

DTIC FILE COPY

Laboratory for Information and Decision Systems  
Massachusetts Institute of Technology  
Cambridge, Massachusetts 02139

86 3729

410450 PTM

86 10 3 036

METHODS OF MULTIPLE ACCESS COMMUNICATIONS

WITH ENERGY DETECTORS



by

Roy D. Yates

B.S.E. Princeton University  
(1983)

SUBMITTED IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS OF  
DEGREE OF

Accession For	1	2	3
1. C. CLASS	<input checked="" type="checkbox"/>		
2. P. P. P.	<input type="checkbox"/>		
3. M. M. M.	<input type="checkbox"/>		
4. T. T. T.	<input checked="" type="checkbox"/>		
By			
Distribution/			
Availability: Copies			
Avail and/or			
Print or Special			
A-1			

MASTER OF SCIENCE

IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1986

Signature of Author Roy D. Yates  
Department of Electrical Engineering and Computer Science  
June 2, 1986

Certified by Robert G. Gallager  
Professor Robert G. Gallager  
Thesis Supervisor

Accepted by Arthur C. Smith  
Chairman, Departmental Committee on Graduate Students

METHODS OF MULTIPLE ACCESS COMMUNICATIONS

WITH ENERGY DETECTORS

by

Roy D. Yates

Submitted to the Department of Electrical Engineering  
and Computer Science on May 9, 1986 in partial fulfillment  
of the requirements for the Degree of Master of Science  
in Electrical Engineering and Computer Science

**ABSTRACT**

For the infinite user population multiple access communication channel in which the number of colliding packets is known exactly, it has been shown previously that it is possible to achieve a throughput arbitrarily close to one. This result is examined.

A particular two-step problem formulation is described. In this strategy, the time axis is divided into many small non-overlapping segments. By enabling subsets of this set of segments, collisions are generated to learn the number of data packets in each segment. In the second stage, each segment found to contain one or more data packets is resolved by splitting.

It is shown that the problem of identifying the number of packets in each segment given the collision results is NP complete. A tradeoff between complexity and throughput is described. The number of packets in the backlog is lower bounded in terms of attainable throughput. The bound indicates that achieving high throughput requires an enormous amount of computation and very large delay, indicating that all strategies of this type are essentially useless for obtaining high throughput.

Thesis Supervisor: Robert G. Gallager

Title: Professor of Electrical Engineering

## Introduction

A multiaccess communication channel consists of a group of transmitting stations sending and receiving data packets. The fundamental problem associated with these channels is to find algorithms for the efficient transmission of messages. Of course, to solve this problem it is necessary to specify a particular model of the communication channel. Typically the following assumptions are made:

1. Arrivals of data packets form a Poisson process with an overall arrival rate  $\lambda$  to the set of transmitters.
2. There are an infinite number of stations. So, by our first assumption, this restriction implies that over a finite lifetime, a particular station will send no more than one packet. This precludes the use of time division multiplexing and focuses attention on algorithms which are efficient when the arrival rate to any one station is small. Moreover, the performance achieved in the infinite station case is a lower bound to the performance in the finite case. With a finite number of stations, when a station has more than one packet waiting for transmission it can pretend to be two separate stations, one for each packet. The effective system is identical to that of an infinite number of stations.
3. The system operates in units of time slots. Each data packet has a length of one slot and every transmission

falls exactly within a single slot. Realistically, a data packet must be slightly shorter than a slot. We will ignore this fact since it has no effect on the types of algorithms used. Also, precise synchronization of station clocks is necessary. This constraint is not particularly difficult to satisfy and will not be considered further.

4. Feedback is instantly received at the end of each slot. In particular, ternary feedback of the following sort is assumed.

- i) If no packets are sent in a slot, then all stations identify that the channel was idle.
- ii) If exactly one station sends a packet in a slot, then all stations correctly receive that packet and identify that there was a successful transmission.
- iii) If two or more stations transmit packets in a slot, then a collision occurs. No data is received, though all stations learn that a collision has occurred.

Thus, in our model, stations attempt to send their packets based on the results of the previous slots. The best of these collision resolution strategies is known as splitting, which we describe by the algorithm below. Note that in this algorithm, enable( $t, \delta$ ) is a function which enables the interval  $[t, t+\delta]$  of the arrival axis and returns the corresponding feedback. That is to say, all packets that were created in the time interval  $[t, t+\delta]$  are transmitted and we learn whether there were zero, one, or more than one packet in the interval. Also, we define  $t$  such that we know that all packets that arrived before time  $t$

have been successfully transmitted. The backlog of the system is simply  $T-t$ , where  $[t,T)$  is the unexamined arrival interval. The algorithm begins by enabling an unexamined arrival interval  $[t,t+\delta_0)$ . The parameter  $\delta_0$  will be chosen to maximize throughput. Lastly, this algorithm assumes there is sufficient backlog such that an optimal interval of length  $\delta_0$  can be enabled. Clearly, when there is no backlog, an interval of length less than  $\delta_0$  would be enabled. Of course, when there is no backlog, enabling an optimum size interval is not very important.

The Splitting Algorithm

```
Begin
  δ := δ₀;
  split := false;
  repeat
    case enable(t,δ) of
      0 : begin
        t := t + δ;
        if split then δ := δ/2;
      end;
      1 : begin
        t := t + δ;
        if not split then δ := δ₀;
          else split := false;
      end;
      c : begin
        δ := δ/2;
        split := true;
      end;
    end; {case}
  until termination;
End.
```

The essential idea behind the algorithm is that whenever there is a collision, we split the enabled interval into two half-intervals each of which can be resolved separately. The above algorithm includes several improvements to this basic idea. If an interval is split into two half-intervals and the first of these half-intervals is found to be

empty, then we know that the second half-interval must contain at least two packets. Consequently, we immediately split the second half-interval, avoiding the certain collision. The second improvement results from the following observation. Suppose an enabled interval contains a collision. In addition, when we split the interval and enable the first half, we find another collision. Given this circumstance, the conditional distribution of packets in the second half-interval is identical to that of an unexamined interval. Resolving this effectively unexamined half-interval is not optimal since this is equivalent to beginning the algorithm with an unexamined interval of length less than the optimal  $\delta_0$ . Instead, we treat the half-interval as part of the unexamined arrival axis and enable a full interval of size  $\delta_0$ .

We define a collision resolution interval as the time from when we enable an interval of length  $\delta_0$  until the next time we enable an interval of length  $\delta_0$ . We define the throughput as the ratio of the expected number of successful transmissions to the expected number of slots used per collision resolution interval. An analysis of this method can be found in [2]. This algorithm remains stable for an arrival rate  $\lambda \leq .4871$  packets/slot. In other words, our maximum possible throughput is .4871. This is achieved by choosing  $\delta_0$  such that  $\lambda\delta_0 = 1.26$  packets. Interestingly, it has been shown [1] that always splitting intervals precisely in half is not optimal. When our splits are made optimally, the maximum throughput becomes .4878 packets/slot. That is, for  $\lambda \leq .4871$ , the average service rate in a collision resolution interval is greater than the arrival rate. Lastly, it has been shown that stable throughputs of greater than .58 packets/slot are not possible [4].

Surprisingly though, if we modify our feedback assumption to be the

following, very different results are possible.

4'. If two or more stations transmit packets in a slot, then a collision occurs. No data is received, though all stations immediately learn exactly how many packets were involved in the collision.

In systems of this type, each station measures the signal energy in every slot. This measurement indicates the number of active senders. We will not consider the feasibility of this assumption, although it is an issue. Obviously, the possibility of inaccurate feedback can only degrade performance.

For systems of this type, Pippenger [5] has demonstrated that throughput arbitrarily close to one is possible. The improvement in achievable throughput, particularly in comparison to ternary feedback, encourages careful consideration of this result. Thus, how to identify the distribution of backlogged packets by the results of collisions will be formulated as a standard problem. We will study the effectiveness of an obvious solution strategy. In addition we will examine the difficulty of our problem and will show a tradeoff between complexity and attainable throughput.

### Problem Formulation, Implementation and Difficulty

Before explaining Pippenger's method, we can indicate why such a result is possible. Suppose we have a very large backlog of packets which have yet to contend for transmission. Based on the time of creation of each contending packet, the backlog can be viewed as a queue over an interval of the time axis. We divide this time interval into L disjoint segments of equal size and define  $x_i$  as the number of packets in segment i. Our problem then is to identify the vector

$$x = [x_1 \dots x_L].$$

Since the arrival process is Poisson, this vector can be viewed as a sequence of L letters from a discrete memoryless source. We wish to encode this sequence for transmission to a destination using a code alphabet consisting of the results of collisions. By enabling a subset of the components of x, that is by instructing all stations with packets in this subset to transmit those packets, we can generate a collision. The destination, which is just the set of stations, learns the number of packets involved in the collision and which segments were allocated. This can be viewed as receiving a code letter. Our intention is to find a mapping of source letters into code letters so as to be able to identify the sequence of source letters by the reception of code letters. This is just a source coding problem. The set of components  $\{x_i\}$  are independent Poisson random variables each with entropy

$$H(U) = - \sum_{k=0}^{\infty} p_k \ln p_k$$

where

$$p_k = \beta^k e^{-\beta} / k!$$

and  $\beta$  is the expected number of packets in each segment. It is readily shown that

$$H(U) \leq \beta^2/2 + \beta - \beta \ln \beta$$

which is constant for fixed  $\beta$ . The source coding theorem then tells us that it is possible to generate a code such that  $\bar{n}$ , the average length of the code words per source letter, obeys

$$\frac{H(U)}{\ln N} \leq \bar{n} < \frac{H(U)}{\ln N} + \frac{1}{L}$$

where  $N$  is the size of the code alphabet. So,  $N$  is simply the number of packets in the backlog. Note that as  $L$  approaches infinity,  $N$  approaches infinity implying that  $\bar{n}$  goes to zero, suggesting that there exist codes for which arbitrarily high throughputs are possible.

However, this observation does not show that there exists such a code that can be implemented by data packet collisions created by the group of stations acting in concert. The difficulty is finding a suitable assignment of source sequences to code words because our set of code words is restricted to be the results of packet collisions. Our code would be described as an algorithm executed locally at each station. This algorithm would indicate which segments will be enabled in a slot given the results of the previous slots.

However, Pippenger shows that we can expect a properly chosen random code to be suitable. To identify  $[x_1 \dots x_L]$ , random collisions are generated to gain sufficient information. In particular, to identify the total number of contending packets, the entire interval is initially allocated; that is, all stations with packets attempt to transmit. From this information,  $L$ , the number of equal size disjoint segments can be chosen appropriately. Pippenger shows that it is possible to identify the number of contending packets in each of these

segments, that is the vector  $[x_1 \dots x_L]$ , in  $K = O(L/\log L)$  number of steps by allocating each segment with probability  $1/2$  in every slot. After identifying the vector  $x$ , a second stage is necessary. In this stage, those segments which hold more than one packet are resolved by executing the splitting algorithm on each separate segment. The reason this scheme achieves high throughput is that in the limit of large  $L$ , the ratio  $K/L$  approaches zero and the number of slots in the second stage necessary to send the  $N$  packets approaches  $N$ . Although this method seems quite reasonable, Pippenger doesn't suggest any method to determine this distribution of packets in the segments from the results of the random allocation process.

The problem of identifying this distribution can be set up in the following way. Once again, the segments of the time axis are numbered 1 through  $L$ . Representing the number of contending packets in segment  $i$  by  $x_i$ , the  $L$  dimensional vector  $x$  completely describes the distribution of packets in the segments. If we have tested  $K$  different subsets with  $y_j$  packets in the  $j^{\text{th}}$  subset then the results of the various collisions are completely described by

$$Ax = y$$

where  $A$  is an  $K \times L$  dimensional 0,1 matrix with  $K < L$  such that  $a_{ij} = 1$  implies that segment  $j$  was enabled during collision  $i$ . Our problem then becomes:

Find  $x$  such that

$$Ax = y$$

$$x, y \geq 0$$

$$x, y \text{ integer}$$

$$A : K \times L \text{ dimensional 0,1 matrix with } K < L.$$

We will call this the identification problem. The A matrix in this formulation is the result of an algorithm executed locally by each station. This algorithm decides which segments to enable in each slot. Of course, these decisions may be based on the outcomes of the earlier collisions. This formulation admits several interpretations. First is the distributed source coding problem described above. A second view is that this is just an integer linear program without an objective function. Note, however, that there may be some advantage in using a particular objective function. Lastly, this formulation can be viewed as an estimation problem. Through an observation matrix A, we observe a vector y. From this, we would like to deduce the vector x, which has a known a priori probability distribution. For the remainder of the discussion, we will use whichever problem description happens to be most convenient in any particular instance.

Since A can be nearly any 0,1 matrix, we will show that the recognition version of the above problem is NP complete. The recognition version of the problem is:

Does there exist x such that

$$Ax = y$$

$$x, y \geq 0$$

$$x, y \text{ integer}$$

$$A : K \times L \text{ dimensional } 0,1 \text{ matrix with } K < L.$$

We will first show that a restricted version of our recognition problem is NP complete.

Theorem: The following problem is NP complete.

Does there exist  $x$  such that

$$Ax = y$$

$$x \in \{0,1\}^L$$

$$y \geq 0$$

$y$  integer

$$A : K \times L, \text{ such that } a_{ij} \in \{0,1\}$$

Proof: Clearly, this problem is in NP since given any 'yes' instance  $x$ , we can readily check that the constraints are satisfied. The NP complete problem that we will transform to our problem is the satisfiability problem which can be stated in the following way:

Given  $m$  clauses,  $C_1, \dots, C_m$  involving the Boolean variables  $x_1, \dots, x_n$ , is the formula  $F(x)$  in conjunctive normal form,

$$F(x) = C_1(x) \cdot C_2(x) \cdot \dots \cdot C_m(x) \text{ satisfiable.}$$

Let  $c_i$  equal the number of literals in  $C_i$ . Replace every occurrence of  $\bar{x}_j$ , the negation of  $x_j$ , with  $z_j$ . For each clause  $C_i$ , we require that

$$\sum_{x \in C_i} x + \sum_{z \in C_i} z + \sum_{j=1}^{c_i-1} s_{ij} = c_i \quad i = 1 \dots m$$

We also add the constraints

$$x_j + z_j = 1 \quad j = 1 \dots n$$

$$x_j, z_j, s_{ij} \in \{0,1\}$$

These constraints can be written in the desired form

$$Ax = y$$

where  $x$  represents  $[x \ z \ s]^T$ . Since all the variables are restricted to be zero or one, our first set of constraints implies that at least one literal of each clause must be true. Our second set of constraints simply requires that either  $x_j$  or  $\bar{x}_j$ , but not both, must be true. A solution  $[x \ z \ s]$  to the transformed problem gives an  $x$  which satisfies  $F(x)$ . In addition, an  $x$  satisfying  $F(x)$  immediately implies a solution  $[x \ z \ s]$  to our transformed problem with  $z = \bar{x}$  and  $s$  chosen trivially to satisfy the constraints. The transformed problem has  $n + m$  equations and fewer than  $2n(m + 1)$  variables. Hence, we have constructed a polynomial time transformation and the problem is NP complete.

Note that requiring  $x \in \{0,1\}^L$  is a restriction on our original recognition problem which allowed  $x$  to belong to the non-negative integers. Consequently, our original recognition problem is NP complete as well.

Curiously, in the context of our data packet problem, the recognition problem as stated has a trivial solution. That is, since we have enabled the segments to create the collisions described by  $Ax = y$ , we know perfectly well that there is an  $x$  satisfying  $Ax = y$  and so the answer to the recognition problem would always be 'Yes, there is such an  $x$ .' Perhaps then, our identification problem is significantly easier than this recognition problem. As we shall demonstrate, this is not the case.

Theorem: The identification problem is NP complete.

Proof: Suppose the contrary so there exists a polynomial time algorithm for our identification problem which finds a solution  $x$  satisfying  $Ax = y$  but only for those instances which we know beforehand to be 'yes' instances. In this case, the algorithm must find such a solution  $x$  for every instance of the problem which is a 'yes' instance. Of course, the algorithm must not find a solution  $x$  for a 'no' instance of the problem. Consequently, for a 'no' problem instance, either our algorithm returns with 'There is no such  $x$ .' or our algorithm does not terminate. However, our algorithm is assumed to run in polynomial time and so there must exist a polynomial upper bound to the amount of time necessary to identify a solution  $x$ . Thus, if our running time were to exceed this bound, we can conclude that this is a 'no' instance. This implies that our polynomial time algorithm solves the NP complete recognition problem, which is a contradiction.

In short, knowing that an instance of the problem is a 'yes' instance does not make identifying a particular solution any easier. Our task to find a distribution of packets  $x$  such that  $Ax = y$  is as difficult as the recognition problem. Moreover, our proof has shown us that looking for zero - one data packet distributions is no easier than looking for non-negative integer distributions. Also, we can place additional restrictions on the  $A$  matrix and the problem will still be NP complete. For example, suppose we had transformed the 3-satisfiability problem, the satisfiability problem in which each

clause contains three literals. The resulting transformation would have no more than five variables in each row constraint. In terms of our data packet problem, this would correspond to the restriction that no more than five segments can be enabled in any collision. This 'simplified' problem is still NP complete.

The NP completeness result, though disheartening, is somewhat misleading. This analysis has overlooked the fact that we are allowed to choose our A matrix. That is, we can specify which segments are to be enabled in each information gathering collision. There may very well be A matrices with some special structure for which our identification problem becomes relatively easy, yet for which high throughputs are still possible. This point remains unresolved and suggests several other questions.

Is there a sensible way to choose our observations dynamically? Perhaps, given the results of the previous collisions, we can specify the segments to be enabled in the next slot in some beneficial manner. Typically though, what one perceives to be sensible is to divide the problem into several smaller subproblems. However, this corresponds to splitting, which eliminates the possibility of asymptotically high throughput.

An example of this approach would be to use the collision information to perform optimal splitting, which has been considered in [6]. Suppose an interval of the arrival axis, which we will label  $[0,1]$  without loss of generality, is known to contain  $k$  packets. This strategy simply enables an interval  $[0,a]$ ,  $0 < a \leq 1$ . We then learn the number of packets in  $[0,a]$  as well as the number of packets in  $[a,1]$ . Resolving packets in each of these sub-intervals is the same problem as resolving packets in the interval  $[0,1]$ . Moreover, for splitting

algorithms, optimal resolution of the sub-interval is also optimal for the entire interval. Thus, we would continue to split these sub-intervals until all  $k$  packets are transmitted. This strategy is readily analyzed.

Denote  $N(a,b)$  as the number of packets in the time interval  $[a,b)$  and  $r_k$  as the minimum expected number of transmissions necessary to resolve an interval known beforehand to contain  $k$  packets. Clearly,  $r_0 = 0$  since an interval containing zero packets requires no transmissions to resolve. Similarly,  $r_1 = 1$ . Given an interval with two or more packets, and a split  $[0,a)$ , we can write down the following recursion:

$$r_k = \begin{cases} 1 + r_k & \text{if } N(0,a) = 0 \\ 1 + r_{k-1} & \text{if } N(0,a) = 1 \\ 1 + r_j + r_{k-j} & \text{if } N(0,a) = j \geq 2 \end{cases}$$

Clearly, for every value of  $k$ , the number of packets in an interval, there is an optimal split  $[0,a_k)$  and  $r_k$  is a function of  $a_k$ . Denoting

$$\begin{aligned} P_n(j,a) &= \Pr[N(0,a) = j \mid N(0,1) = n] \\ &= \binom{n}{j} a^j (1-a)^{n-j} \end{aligned}$$

we can write

$$\begin{aligned} r_n(a_n) &= [1 + r_n(a_n)]P_n(0,a_n) + [1 + r_{n-1}(a_{n-1})]P_n(1,a_n) \\ &\quad + \sum_{j=2}^n \left[ 1 + r_j(a_j) + r_{n-j}(a_j) \right] P_n(j,a_n) \quad n \geq 2 \end{aligned}$$

Solving for  $r_n(a_n)$ , we find that

$$\begin{aligned} r_n(a_n) &= 1 - P_n(1,a_n) + \frac{\sum_{j=0}^{n-1} r_j(a_j) \left[ P_n(j,a_n) + P_n(n-j,a_n) \right]}{1 - P_n(n,a_n) + P_n(0,a_n)} \end{aligned}$$

Given  $\{r_0, \dots, r_{n-1}\}$  and  $\{a_0, \dots, a_{n-1}\}$ , we can minimize over  $a_n$  to find  $r_n$ . In particular, the first few terms are:

$n$	$r_n$	$a_n$
2	3.000	0.500
3	4.787	0.412
4	6.632	0.343
5	8.485	0.288
6	10.342	0.249
7	12.203	0.220
8	14.067	0.197

Now, suppose we have an unexamined interval  $[a, b]$ . First, we would enable the entire interval to find  $k$ , the total number of packets. If  $k \leq 1$ , then we are all done. Otherwise, we would expect to need an additional  $r_k$  transmissions to send all of the packets. So, our expected number of transmissions would be

$$R_k = \begin{cases} 1 & \text{if } k \leq 1 \\ 1 + r_k & \text{if } k > 1 \end{cases}$$

We can define our throughput as the ratio  $E[k]/E[R_k]$ . Note that  $E[k] = \lambda(b - a)$  where the time difference  $(b - a)$  is measured in slots. All that remains is to choose  $(b - a)$  to maximize our throughput. It turns out that choosing our arrival interval so that expected number of packets equals 1.88 is optimal. The resulting throughput achieved is .5316 packets/slot.

This is a modest improvement over standard splitting algorithms using ternary feedback. Why these systems do not perform better can be easily identified. Suppose in the interval  $[0, 1]$  we know there are  $k$  packets. Then, we know that

$$\Pr[N(0, a) = 1 \mid N(0, 1) = k] = ka(1 - a)^{k-1} \quad 0 \leq a \leq 1$$

This probability is maximized by choosing

$$a = 1/k$$

yielding

$$P(k) = (1 - 1/k)^{k-1}$$

Note that

<u>k</u>	<u>P(k)</u>
1	1
2	.5
3	.44
4	.42

and that

$$\lim_{k \rightarrow \infty} P(k) = e^{-1}$$

Except when there is a single packet in the interval, the probability of a successful transmission is always less than one half. Essentially, there is no benefit in waiting for a large backlog. These strategies serve only to avoid wasting slots in those instances when an unexpectedly large number of packets arrives in some interval. In fact, suppose the feedback were restricted so that more than N packets in a collision is identified as such, rather than by the explicit number of packets involved. In the analysis [6], it has been shown that for  $N \geq 5$ , there is only a negligible loss in throughput using this limited feedback. This is not surprising, since the optimal solution of this type looks at intervals which are expected to contain relatively few packets. Moreover, this emphasizes the importance of generating large collisions.

### Throughput versus Complexity

Another approach to identifying the packet distribution from the collision results is to simply conduct a search for the distribution which satisfies the collision requirements. If it were possible to achieve reasonably high throughputs without having to construct an extraordinarily large problem, then searching becomes a viable technique. Of course, imposing some special structure on our observation matrix may allow reduced search effort for equivalent effect. However, rather than address the question of special structures, we will concentrate on the issue of how large a problem we must consider in order to attain a specified throughput, irrespective of any special structure.

The size of our problem is a function of the number of packets in the interval we are considering. Clearly, a smaller number of packets can distribute themselves in a smaller number of ways, resulting in a simpler search. So, our intention will be to lower bound the number of packets required in the backlog in order to achieve a certain throughput. Pippenger's proof showed that we can obtain high throughputs by considering very large problems. However, the possibility that modest sized formulations might do reasonably well was not eliminated. This issue will now be considered.

We will assume we have an interval containing  $N$  packets. This involves no loss of generality. Since knowledge of  $N$  can only increase our throughput, any upper bound to throughput under this assumption is still a valid upper bound when  $N$  is unknown. Moreover, if we were to implement a collision resolution algorithm, it seems likely that enabling the entire interval expressly to learn  $N$  would be our first

step. We then divide our large interval into L small segments. L is a parameter to be optimized later. L will be a function of N although at this point, it is not clear what will be an appropriate choice. Define  $\mathbf{x}$  to be the L dimensional vector such that  $x_i$  represents the number of packets in segment i and  $\sum x_i = N$ .

Our collision resolution algorithm will have two steps. First, we will generate collisions of the form

$$\mathbf{y} = \mathbf{Ax}$$

where A is a  $K \times L$  dimensional matrix such that  $a_{ij} \in \{0,1\}$ . These K collisions should serve to identify the vector  $[x_1 \dots x_L]$ . Second, every segment i containing  $x_i > 0$  packets, we will resolve separately by splitting. If this second step uses J slots to transmit successfully the N packets described by  $\mathbf{x}$ , then our throughput will be

$$t = \frac{\text{Number of packets}}{\text{Expected number of transmissions}}$$

$$t = \frac{N}{K + J}$$

We will find lower bounds for K and J, so we can upper bound throughput.

Our bounding arguments will assume that no packets are successfully transmitted during the first stage. This assumption implies that the A matrix must not be sparse. In our earlier coding argument, we showed that we need our alphabet of output 'code letters' to be large. If A were sparse, then our output  $\mathbf{y}$  would consist of small integers and the size of our alphabet would be small. Thus, our assumption is reasonable. Moreover, any strategy designed to achieve high throughput must generate collisions involving large number of transmitters. Consequently, every high throughput algorithm can be described by this two step approach, where step one generates large collisions and step

two transmits packets with very few collisions.

We will use the Source Coding Theorem to derive a lower bound for  $K$ .  
The Source Coding Theorem says [3]:

Let a discrete memoryless source have finite entropy  $H(X)$  and consider a coding from sequences of  $L$  source letters into sequences of  $K$  code letters from a code alphabet of size  $D$ . Only one source sequence can be assigned to each code sequence and we let  $P_e$  be the probability of occurrence of a source sequence for which no code sequence has been provided. Then, for any  $\delta > 0$  if

$$K/L \geq [H(X) + \delta]/\log D$$

$P_e$  can be made arbitrarily small by making  $L$  sufficiently large. Conversely, if

$$K/L \leq [H(X) - \delta]/\log D$$

then  $P_e$  must become arbitrarily close to 1 as  $L$  is made sufficiently large.

We can view the sequence  $\{y_1 \dots y_K\}$  as a sequence of  $K$  code letters such that  $y_i \in \{0, 1, \dots, D-1\}$ . Since our arrivals are Poisson and our segments do not overlap, we note that  $\{x_i | i = 1..L\}$  can be viewed as a set of  $L$  independent source letters. The entropy of this sequence is

$$\begin{aligned} LH(X) &= H(X_1 \dots X_L) \\ &= H(X^L) \end{aligned}$$

Since conditioning always reduces entropy, we know that

$$H(X^L) \geq H(X^L|N)$$

where  $H(X^L|N)$  denotes the entropy of the sequence  $x_1 \dots x_L$  conditional on

$$\sum_{i=1}^L x_i = N$$

Consequently, the second part of the Source Coding Theorem implies that if

$$K \log D < H(X^L|N)$$

then  $P_e$  must approach 1 as L becomes sufficiently large. We will call this the entropy bound. That is, if the entropy bound holds, then the probability of occurrence of a source sequence for which no code sequence has been provided approaches one for L large. In the context of our original problem this allows us to make the following argument. Suppose we generate a particular collision matrix A and to every vector  $x$  in the set

$$X = \{x \mid \sum x_i = N, 0 \leq x_i \leq N\}$$

we assign the code word  $y = Ax$ , if the code word  $y$  has not been assigned previously. Otherwise, if  $y$  has already been assigned, then  $x$  is labelled unassigned. So, if the entropy bound is true, then the probability of occurrence of an unassigned source sequence  $x$  approaches one for L sufficiently large. Equivalently, we can say the probability of occurrence of a vector  $y$  for which there exists more than one vector  $x \in X$  such that  $y = Ax$ , approaches one for large L. In short, we have the following result:

**Theorem:** If

$$K \log D < H(X^L|N)$$

then the probability that we will not be able to decode  $x$  given  $y = Ax$  approaches 1 for L sufficiently large.

To make use of this claim, we must identify D and  $H(X^L|N)$ . To find

the code alphabet size D, we observe that

$$y_i = \sum_{j=1}^L a_{ij} x_j \leq \sum_{j=1}^L x_j = N$$

So,

$$y_i \in \{0, \dots, N\}$$

which implies

$$D = (N + 1)$$

To evaluate  $H(X^L|N)$  is more difficult. Define the length of our initial enabled interval to be  $\mu$  slots long ( $\mu$  is not usually an integer) and

$$\begin{aligned} P[X^L|N] &= \Pr[x_1=n_1 \dots x_L=n_L \mid \sum x_i = N] \\ &= \Pr[x_1=n_1 \dots x_L=n_L, \sum x_i = N] / \Pr[\sum x_i = N] \\ &= \frac{(\lambda\mu/L)^{n_1} e^{-(\lambda\mu/L)} / n_1! \cdot \dots \cdot (\lambda\mu/L)^{n_L} e^{-(\lambda\mu/L)} / n_L!}{(\lambda\mu)^N e^{-\lambda\mu}/N!} \\ &= \frac{N!}{L^N (n_1! \cdot \dots \cdot n_L!)} \end{aligned}$$

The entropy becomes

$$\begin{aligned} H(X^L|N) &= - \sum_{n_1 \dots n_L} \left[ \frac{N!}{L^N (n_1! \dots n_L!)} \right] \log \left[ \frac{N!}{L^N (n_1! \dots n_L!)} \right] \\ &= N \log L - \log(N!) + E[\ln n_j]; \end{aligned}$$

Since  $n_j! \geq 1$ , for all j,

$$E[\ln n_j] \geq 0$$

In addition,

$$N! \leq N^N$$

implying that

$$H(X^L|N) \geq N \log(L/N)$$

Suppose we are now generating collisions using some A matrix. For

every source sequence  $x^L$ , we generate  $K$  collisions and we know that in order to decode  $x$  from  $y = Ax$  every time, we must have

$$K \geq H(x^L|N) / (\log D)$$

$$\geq N \log(L/N) / \log(N+1)$$

We now can evaluate the second stage of our collision resolution process. Suppose we are handed a particular distribution of  $N$  packets into our  $L$  segments,  $[x_1 \dots x_L]$ . Segments containing a single packet are transmitted directly. Segments containing more than one packet will be resolved using the optimal splitting algorithm analyzed earlier.

We would like to determine  $J$ , the expected number of transmissions necessary to send our  $N$  packets given the packet distribution  $x$ . Denote  $r_k$  as the expected number of transmissions needed to resolve a particular segment given that we have already identified that there are  $k$  packets in the segment. Happily, is this the same  $r_k$  that we determined earlier. Given the distribution  $x$ , the number of slots necessary to resolve a particular segment is independent of the number of slots necessary to resolve any other segment. Also the number of packets in a particular segment is binomially distributed and is described by:

$$P_N(k) = \Pr[x_j = k \mid \sum x_i = N]$$

$$= \frac{\Pr\left[x_j = k, \sum_{i \neq j} x_i = N-k\right]}{\Pr[\sum x_i = N]}$$

$$= \frac{[(\lambda\mu/L)^k e^{-(\lambda\mu/L)/k!}] [(\lambda\mu(1 - 1/L))^{N-k} e^{-\lambda\mu(1 - 1/L)/(N-k)!}]}{(\lambda\mu)^N e^{-\lambda\mu} / N!}$$

$$P_N(k) = \binom{N}{k} (1/L)^k (1 - 1/L)^{N-k}$$

These facts allow us to write

$$J = L \sum_{k=1}^N r_k P_N(k)$$

The values  $r_k$  are summarized here:

$$r_0 = 0$$

$$r_1 = 1$$

$$r_2 = 3$$

$$r_k \geq 1.874k - 1 \quad \text{for all } k$$

The inequality for  $k \geq 3$  is very tight.

Applying this result, we have

$$J \geq L \left[ \sum_{k=0}^N (1.87k - 1)P_N(k) + P_N(0) + .13P_N(1) \right]$$

or

$$J \geq L \left[ 1.87(N/L) - 1 + (1 - 1/L)^N + .13N(1/L)(1 - 1/L)^{N-1} \right]$$

We will define  $a$  such that  $L = aN$ , yielding

$$J/N \geq 1.87 - a + a(1 - 1/aN)^N + .13(1 - 1/aN)^{N-1}$$

Asymptotically, this leads to the following definition

$$\begin{aligned} r(a) &= \lim_{N \rightarrow \infty} \left[ 1.87 - a + a(1 - 1/aN)^N + .13(1 - 1/aN)^{N-1} \right] \\ &= 1.87 - a + (a + .13)e^{-1/a} \end{aligned}$$

For large values of  $N$ , given  $L/N = a$ , we would expect to need  $Nr(a)$  slots to send successfully all  $N$  data packets. The function  $r(a)$  is convex and non-increasing. Suppose, we were able to determine the distribution  $[x_1 \dots x_L]$  without generating a single collision. In this unlikely circumstance, our throughput for  $a \leq 1$ , would be

$$t = N/J \leq 1/r(a)$$

If we chose  $a = 1$ , that is  $L = N$ , we would have  $t \leq 1/r(1) = .74$ . To achieve a particular throughput, we must choose  $a$  sufficiently large so that the likelihood of collisions in the second stage is suitably small. Obviously, since we normally must identify  $[x_1 \dots x_L]$ , which is not an easy task, our actual attainable throughput must be significantly lower.

In addition, note that

$$\lim_{a \rightarrow \infty} r(a) = 1$$

For  $L$  very much larger than  $N$ , only  $N$  slots are necessary to transmit all  $N$  packets. Given the distribution  $x$ , we see that the likelihood that a single segment contains more than one packet becomes very small for large  $L$ . This indicates that our bound on  $J$  is asymptotically tight.

We now can combine the results of our two stages.

$$\begin{aligned} t &= \frac{N}{K + J} \\ &= \frac{1}{\frac{K/N}{N} + r(a)} \end{aligned}$$

Using our lower bound for  $K$ , we have

$$t \leq \left[ \frac{\log a}{\log N+1} + r(a) \right]^{-1}$$

We can rearrange this expression to yield

$$\ln N+1 \geq [\log a] \left[ \frac{t}{1 - r(a)t} \right]$$

For every choice of  $t$ , there is a value of  $a$  which minimizes this lower bound. Solving this minimization problem for each value of  $t$  yields a lower bound on  $N$ . The results of this minimization process are displayed in Figure 1.

For  $t$  close to one, we can continue to lower bound  $N$ , giving an indication of its asymptotic growth. Substituting our formula for  $r(a)$  into our above bound, we have

$$t \leq \left[ \frac{\ln a}{\ln N+1} + 1.87 - a + (a + .13)e^{-1/a} \right]^{-1}$$

Noting that

$$e^{-1/a} > 1 - 1/a + 1/2a^2 - 1/6a^3$$

we learn

$$t \leq \left[ \frac{\ln a}{\ln N+1} + 1 + .37/a - .232/a^2 \right]^{-1}$$

Suppose we choose  $a^*$  such that

$$1 + .37/a^* + .232/(a^*)^2 = 1/t$$

Solving this equation for  $a^*$ , we have for  $t \geq .87$

$$a^* = .185\rho[1 + (1/2)(1 - 6.78/\rho)^{1/2}]$$

where  $\rho = t/(1-t)$ . Note that in order to achieve a throughput  $t$ , we must have  $a \geq a^*$ . Since  $1/t \geq 1$ , we can say

$$t \leq \left[ \frac{\log a^*}{\log N+1} + 1 \right]^{-1}$$

This implies

$$\begin{aligned} N+1 &\geq e^{(\ln a^*)\rho} \\ &\geq e^{\rho \ln (.185\rho)} \\ &= [\rho/5.4]^{\rho} \quad \rho = \frac{t}{1-t} \quad t \geq .87 \end{aligned}$$

Along with figure 1, this describes the tradeoff between complexity and high throughput for this problem. However it is a very discouraging tradeoff. For example, we note from figure 1, that to obtain a throughput of .95, we must consider a backlog containing at least  $e^{84}$

packets. Moreover, there is no guarantee that considering such a large number of packets will yield the desired throughput. When we remind ourselves that the underlying stage one problem to identify the distribution  $x$  is NP complete, the size of the problem generated seems unusually daunting. In particular, we can evaluate the size of the set over which we must search. If we are simply looking for solutions  $x$  to  $y = Ax$ , then the search time would be proportional to the size of the set

$$X = \{x : \sum x_i = N\}$$

We can lower bound the cardinality of this set.

$$|X| = \sum_{R=1}^N \binom{L}{R} \binom{N-1}{R-1}$$

To verify this claim, note that  $\binom{L}{R}$  is the number of ways of choosing  $R$  components of  $x$  to be nonzero. Also,  $\binom{N-1}{R-1}$  equals the number of ways to distribute  $N$  packets in the  $R$  nonzero components  $x_{k_1}, \dots, x_{k_R}$ . To see this, observe that we can equivalently specify the sequence

$$x_{k_1} \cdot x_{k_1} + x_{k_2} \cdot \dots \cdot x_{k_1} + \dots + x_{k_R}$$

This is a strictly increasing sequence such that

$$x_{k_1} + \dots + x_{k_R} = N$$

However, the first  $R - 1$  terms of the sequence can be chosen without repetition from the set  $\{1, \dots, N - 1\}$  in  $\binom{N-1}{R-1}$  possible ways.

Note that

$$\binom{N-1}{R-1} = (R/N) \binom{N}{R} \geq (1/N) \binom{N}{R}$$

As a result,

$$|x| \geq (1/N) \sum_{R=1}^N \frac{L}{R} \frac{N}{R} = (1/N) \binom{L+N}{N}$$

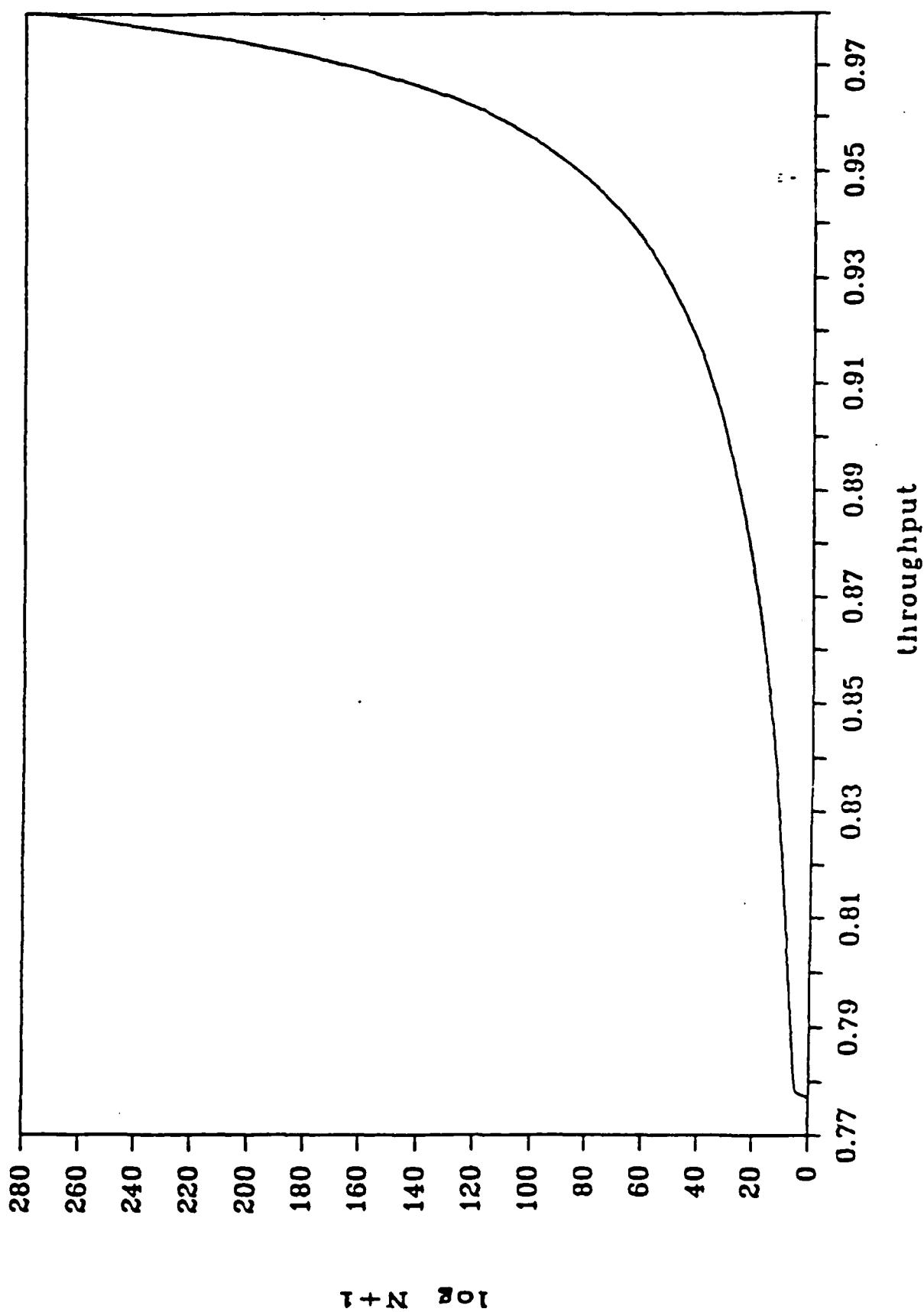
$$= (1/N) \binom{(1+\alpha)N}{N}$$

When we remind ourselves of how rapidly N diverges, we have shown that any search strategy that achieves high throughput would require a truly enormous number of computations. Even if we were to find a special observation matrix that makes solving the stage one problem relatively easy, we would still have an extraordinarily large 'easy' problem. In addition, no such special matrix has yet to be identified.

### Conclusion

Our lower bound on the number of packets is valid only for strategies that can be described by our problem formulation. In particular, collision resolution does not require identifying the number of packets in every small segment of the arrival time axis. All that is truly necessary is to be able to specify subsets of the arrival axis that contain a single data packet. Strategies which solve this problem without reverting to the  $Ax = y$  segmented problem formulation have yet to be described. Moreover, for any approach that yields high throughput, it would appear from our source coding arguments that a large code alphabet is necessary. In other words, we must generate collisions involving large numbers of data packets. Although precisely how large is an issue, it seems clear that some tradeoff between complexity and throughput would still exist. In short, our results suggest that without doing an extraordinary amount of work, we cannot hope to do significantly better than the throughput of .53 attained by the optimal splitting algorithm.

Figure 1



REFERENCES

- [1] Mosely, J., 'An Efficient Contention Resolution Algorithm for Multiple Access Channels', M.S. thesis, MIT, Tech. Rep. LIDS-TH-918.
- [2] Gallager, R. G., 'A Perspective on Multiaccess Channels', IEEE Transactions on Information Theory, March, 1985.
- [3] Gallager, R. G., Information Theory and Reliable Communication, John Wiley, 1968.
- [4] Tsybakov, B. S. and Mikhailov, V. A., 'An Upper Bound to Capacity of Random Multiple Access Systems', presented at the Information Theory Symposium, Santa Monica, CA, 1981; also Probl. Peredach. Inform., Vol 17, No. 1, pp. 90-95, January - March, 1981.
- [5] Pippenger, N., 'Bounds on the Performance of Protocols for a Multiple Access Broadcast Channel', IEEE Transactions on Information Theory, Vol. IT-27, pp. 145-151, March, 1981.
- [6] Georgopoulos, M. and Papantoni-Kazakos, P., 'Collision Resolution Protocols Utilizing Absorption and Collision Multiplicities', Dept. of Elec. Eng. and Comput. Sci., Univ. of Connecticut, Storrs, Tech. Rep. TR-83-5, April 1983.

E V D

11-86

DTC